Uma Metodologia Baseada em Simulação e Algoritmo Genético para Exploração de Estratégias de Escalonamento de Laços

Pedro H. Penna[1], **Márcio Castro**[2], **Henrique C. Freitas**[1], **François Broquedis**[3], **Jean-François Méhaut**[3]

[1]Departamento de Computação – Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte – Brasil

[2]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – Brasil

[3]CEA-DRT – Universidade de Grenoble Alpes
Saint-Martin d'Hères – França

pedro.penna@sga.pucminas.br, marcio.castro@ufsc.br, cota@pucminas.br,
francois.broquedis@imag.fr, jean-francois.mehaut@imag.fr

Abstract. *In High Performance Computing, the application's workload must be well balanced among the threads to achieve better performance. In this work, we propose a methodology that enables the design and exploration of new loop scheduling strategies. In this methodology, a simulator is used to evaluate the most relevant existing scheduling strategies, and a genetic algorithm is employed to explore the solution space of the problem itself. The proposed methodology allowed us to design a new loop scheduling strategy, which showed to be up to 32.3x better than the existing policies in terms of load balance.*

Resumo. *Na Computação de Alto Desempenho, a carga de trabalho de aplicações paralelas deve ser balanceada entre as threads para obtenção de um melhor desempenho. Nesse trabalho é proposta uma metodologia que possibilita o projeto e a exploração de novas políticas de escalonamento de laços. Essa metodologia emprega a técnica de simulação para estudar as principais estratégias existentes e um algoritmo genético para explorar o espaço de soluções do problema. A metodologia proposta auxiliou no projeto de uma nova estratégia de escalonamento de laços que se mostrou ser até 32.3x melhor em ternos de balanceamento de carga que as políticas existentes.*

1. Introdução

De maneira geral, aplicações podem ser classificadas em dois grupos: regulares, que se caracterizam pela homogeneidade no tamanho de suas tarefas; e irregulares, que se destacam pela sua heterogeneidade. A multiplicação de matrizes é um exemplo de aplicação regular, onde as tarefas possuem cargas similares e consistem na multiplicação de uma linha de uma matriz pela coluna de outra. Em contrapartida, uma aplicação de ordenação de inteiros exemplifica a classe de aplicações irregulares, com o número de operações necessárias para se ordenar um número variando em função dos demais.

A diferença existente entre essas duas classes é fundamental para o projeto de aplicações paralelas eficientes. Nesse contexto, aplicações regulares são fortemente apreciadas, pois uma simples divisão de uma carga de trabalho total n entre t *threads* é suficiente para se obter uma distribuição de tarefas homogênea. Infelizmente, essa estratégia não é válida para aplicações irregulares, uma vez que as tarefas são heterogêneas em tamanho. Logo, um determinado conjunto de *threads* poderá apresentar um forte desbalanceamento de carga, fazendo com que o desempenho da aplicação fique dominado pela *thread* mais sobrecarregada.

O problema de balanceamento de carga pode ser estudado em diferentes granularidades [Patni et al. 2011, Tantawi and Towsley 1985]. Porém, esse problema é estudado no presente trabalho no contexto de escalonamento de laços (*loop scheduling*), onde as iterações são consideradas como tarefas que devem ser atribuídas às *threads*. Nessa linha, diferentes estratégias são conhecidas para se endereçar esse problema [Srivastava et al. 2012]. O objetivo do presente trabalho, no entanto, é apresentar uma metodologia que possibilite o projeto e a exploração de novas estratégias de escalonamento de laços, em especial aquelas que levem em consideração a característica da carga de trabalho da aplicação. A metodologia proposta utiliza a técnica de simulação para estudar as principais políticas de escalonamento existentes e um algoritmo genético para explorar o espaço de soluções do problema. Por fim, uma nova estratégia, projetada com base nos resultados obtidos com a aplicação da metodologia, é proposta para fins de validação. Essa estratégia, nomeada *Smart Round-Robin* (SRR), considera características da carga de trabalho e, então, acaba por se destacar das estratégias existentes. As principais contribuições desse trabalho em relação ao estado da arte se concentram em: (i) uma proposta da metodologia para estudo de novas estratégias de escalonamento de laços; (ii) no uso de um algoritmo genético aplicado ao contexto; e (iii) uma proposta de uma nova estratégia de escalonamento de laços.

O restante do artigo está organizado da seguinte forma. Na Seção 2 os principais trabalhos relacionados ao problema de escalonamento de laços são apresentados. Na Seção 3 o problema em questão é discutido formalmente. Na Seção 4 a metodologia proposta é apresentada em detalhes. Na Seção 5 os resultados obtidos são apresentados e discutidos. Por fim, na Seção 6, as conclusões e trabalhos futuros são apresentados.

2. Trabalhos Relacionados

O escalonamento de laços é um tema recorrente na área de Computação Paralela e de Alto Desempenho. Nesta seção são discutidos os principais esforços em pesquisa nesse assunto que são relevantes para o presente trabalho.

Diversas estratégias para realizar o escalonamento de laços foram propostas para satisfazer as mais variadas restrições [Hajieskandar and Lotfi 2010, Olivier et al. 2012, Ding et al. 2013, Chen and Guo 2014]. No entanto, selecionar estratégias de escalonamento mais adequadas para aplicações específicas ainda é um desafio. Objetivando solucionar esse problema, Sukhija *et al.* propuseram uma abordagem que utiliza técnicas de aprendizado de máquina supervisionado para predizer e escolher a estratégia de escalonamento de laços mais robusta para uma aplicação e plataforma específicos [Sukhija et al. 2014]. Os resultados mostraram que: (i) a abordagem proposta é capaz de selecionar a estratégia de escalonamento mais robusta, que satisfaça uma tolerância especificada pelo usuário; e (ii) a estratégia selecionada usualmente oferece maior

robustez do que outra selecionada empiricamente para a mesma aplicação.

Seguindo uma linha de pesquisa semelhante, Srivastava *et al.* propuseram e avaliaram uma rede neural artificial capaz de predizer a flexibilidade de estratégias dinâmicas para o escalonamento de laços em sistemas heterogêneos [Srivastava et al. 2013]. O modelo proposto utiliza como parâmetros de entrada a quantidade de processadores e suas respectivas disponibilidades, o número de iterações no laço e a função de probabilidade dos tempos de execução dessas iterações. Três funções de distribuição probabilísticas foram consideradas (*gaussiana*, *gamma* e *exponencial*) e tiveram seus parâmetros variados, totalizando 1.152 exemplos. Os resultados mostraram que o modelo proposto é efetivo na seleção da estratégia de balanceamento mais robusta, garantindo uma acurácia de $94{,}9\,\%$.

No que diz respeito à avaliação das estratégias de escalonamento existentes, Srivastava *et al.* propuseram uma metodologia para analisar a robustez de estratégias de escalonamento dinâmico de laços [Srivastava et al. 2012]. Para tanto, os autores implementaram oito estratégias em um simulador e utilizaram uma aplicação sintética para avaliá-las. Essa aplicação possui n laços independentes e o tempo de execução de cada iteração de um laço é representado por uma distribuição *gaussiana*. Ao final do estudo, os autores concluíram que a metodologia baseada em simulação pode ser aplicada para realizar uma análise das estratégias de balanceamento existentes. Outro trabalho que segue uma metodologia de simulação similar é discutido em [Balasubramaniam et al. 2012].

O presente trabalho se difere dos trabalhos discutidos anteriormente em diferentes aspectos. Em primeiro lugar, ele se concentra em apresentar uma metodologia que possibilite o projeto e a exploração de estratégias de escalonamento de laços, levando em consideração a carga de trabalho da aplicação. Nessa metodologia, a técnica de simulação é empregada para avaliar as principais estratégias de escalonamento existentes e uma técnica de busca é usada para se explorar o espaço de soluções do problema. Em segundo lugar, esse trabalho amplia o estudo de aplicações sintéticas para 5 diferentes distribuições de carga. Finalmente, em terceiro lugar, uma nova estratégia de escalonamento, que se baseia nos resultados da aplicação da técnica de exploração, é apresentada e comparada com as estratégias existentes, validando assim a metodologia proposta.

3. O Problema de Escalonamento de Laços

O problema de escalonamento de laços é uma instância particular do problema de balanceamento de carga de trabalho e consiste em um problema NP-Completo de minimização que é definido formalmente a seguir [Skiena 2008]. Considere um conjunto de n iterações $T = \{t_1, t_2, \ldots, t_n\}$, em que cada iteração t_i está associada a um tamanho s_i, com $s_i \in \mathbb{N}$. Assim, dado um inteiro $k \geq 1$, o problema consiste em particionar T em k subconjuntos disjuntos $\{C_1, C_2, \ldots, C_k\}$ de forma a minimizar a máxima soma $s_{C_i} = \{\sum s_i \mid s_i \in C_i\}$ dentre os conjuntos e, então, reduzir o grau de desbalanceamento entre partições (Equação 1):

$$f(s_{C_i}) = \min\left(\max\left(s_{C_i}\right)\right), \quad \forall\, 1 \leq i \leq k \tag{1}$$

O número de iterações e partições constituem variáveis intrínsecas ao problema de escalonamento de laços e, portanto, não podem ser ignoradas em sua análise. No entanto, quando o objetivo é estudar o problema no contexto de uma aplicação real, outras variáveis também devem ser consideradas, o que torna a função de minimização apresen-

tada multiobjetiva. Nesse segundo cenário, algumas das variáveis que podem ser destacadas são: a carga associada às iterações, a frequência em que o escalonamento é realizado e a afinidade de memória existente entre as iterações.

A carga associada a cada iteração está relacionada à natureza da carga de trabalho da aplicação e exerce influência direta na qualidade da solução para o problema. Por exemplo, se a carga das iterações seguir uma distribuição *uniforme*, isto é, se a probabilidade de ocorrência de iterações com carga muito pequena e muito grande é a mesma, o grau de desbalanceamento (Equação 1) observado tende a ser menor. No entanto, para uma distribuição não tão regular, esse grau tende a ser mais elevado. A frequência com que o escalonamento de iterações é realizado, por sua vez, é importante para algoritmos de escalonamento dinâmico que realizam a atribuição de iterações em tempo de execução. Nesse contexto, ela dita quantas vezes a estratégia de escalonamento será invocada durante a execução da aplicação e, então, impacta diretamente em seu desempenho final. Por fim, a afinidade de memória se relaciona com o nível de localidade espacial e/ou temporal existente entre as iterações atribuídas a uma mesma partição. Quando as localidades espacial e temporal são altas, há uma melhora significativa na eficiência do sistema de memória como um todo, minimizando contenções em barramentos e outras estruturas de interconexão, impactando positivamente no desempenho da aplicação.

4. Metodologia

O presente trabalho tem como principal objetivo apresentar uma metodologia que possibilite o projeto e a exploração de novas estratégias de escalonamento de laços. Nessa metodologia, a técnica de simulação é empregada para estudar as principais políticas de escalonamento existentes e um algoritmo genético é usado para se explorar o espaço de soluções do problema. Nessa seção, essa metodologia é apresentada em detalhes. Inicialmente é feita uma discussão sobre as variáveis consideradas no problema. Em seguida, o ambiente de simulação utilizado é apresentado. Por fim, é apresentado o algoritmo genético empregado nesse trabalho.

4.1. Variáveis Consideradas

Na Seção 3 discutiu-se o problema de escalonamento de laços e as principais variáveis a ele relacionadas. Algumas dessas variáveis são inerentes ao problema, enquanto outras estão relacionadas às características da aplicação ou da plataforma de execução, como a carga associada às iterações e a afinidade de memória, respectivamente. Nesse estudo, as seguintes variáveis foram consideradas: (i) número de iterações; (ii) número de *threads* (partições); e (iii) a função de distribuição de carga das iterações.

As duas primeiras variáveis (número de iterações e *threads*) constituem a natureza do problema de escalonamento de laços, sendo portanto consideradas no estudo. A terceira variável está relacionada à característica da aplicação e dita qual é a carga associada às iterações. Essa variável é também considerada no presente trabalho devido à sua influência no grau de desbalanceamento existente entre as *threads* e ao seu impacto na qualidade da solução proporcionada pela estratégia de escalonamento adotada.

As demais variáveis discutidas na Seção 3 não são consideradas devido a dois motivos principais: (i) ou estão relacionadas a alguma das variáveis já consideradas, como por exemplo a máxima diferença entre a carga de duas iterações quaisquer, que se correlaciona com a função de distribuição de carga das iterações; ou (ii) são dependentes de

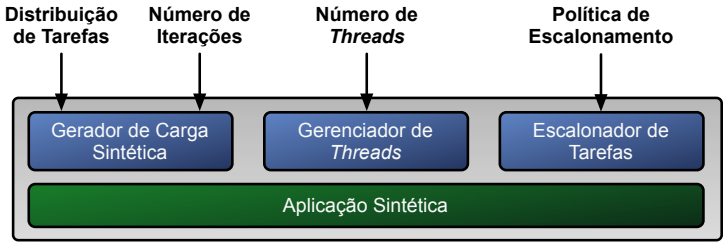


Figura 1. Arquitetura interna do simulador desenvolvido.

plataforma e, portanto, podem ser estudadas em uma etapa posterior, como por exemplo a afinidade de memória existente entre as iterações atribuídas a uma mesma *thread* e sua relação com a hierarquia de memória da plataforma considerada.

4.2. Simulador

Uma vez definidas as variáveis a serem estudadas, um ambiente para estudo das políticas de escalonamento de laços existentes se faz necessário. Para tanto, duas alternativas são possíveis: (i) usar uma plataforma real; ou (ii) usar um ambiente simulado. A primeira alternativa possui como principal vantagem possibilitar uma análise de todas as variáveis envolvidas no problema. No entanto, conforme discutido na Seção 4.1, algumas dessas variáveis são dependentes de plataforma e de aplicação e, uma vez que o presente trabalho se concentra em uma análise teórica do problema, elas podem ser abstraídas. Em contrapartida, o uso de um ambiente simulado possibilita que as variáveis de interesse não só sejam isoladas, como também controladas minuciosamente. Por esse motivo, um simulador foi projetado, desenvolvido e utilizado no presente trabalho. Vale ressaltar que simuladores já existentes reportados em trabalhos relacionados [Srivastava et al. 2012, Balasubramaniam et al. 2012] não foram utilizados por não estarem disponíveis em domínio público e limitarem a análise de cargas sintéticas a apenas 3 distribuições de carga.

A Figura 1 apresenta a arquitetura interna desse simulador, com os seus três módulos principais na parte superior e aplicação sintética representada na parte inferior. O módulo gerador de carga sintética recebe como parâmetros o número de iterações e a função de distribuição da carga associada a essas iterações, e gera como saída um conjunto de iterações com as características especificadas. A carga de uma iteração representa o tempo necessário para que ela seja concluída, de forma que conjunto de iterações corresponde à carga de trabalho da aplicação sintética.

O módulo gerenciador de *threads*, por sua vez, é responsável por criar e escalonar as *threads* da aplicação sintética. O número de *threads* criadas depende do parâmetro especificado e o escalonamento de *threads* é feito segundo uma política *round-robin*. Para gerenciar as *threads*, esse módulo mantém duas estruturas: uma fila de prioridades das *threads* em execução, ordenada pelo tempo de processamento remanescente de cada *thread*; e uma lista das *threads* prontas para execução. Por fim, o módulo escalonador de iterações realiza, segundo a política de escalonamento especificada, o escalonamento das iterações criadas pelo gerador de carga sintética. O escalonador exporta uma interface que possibilita o estudo tanto de estratégias estáticas quanto dinâmicas. Nessa arquitetura, a aplicação sintética é responsável por processar linearmente o conjunto de iterações gerado, considerando os parâmetros especificados.

Esse simulador está disponível *online*[1], o que permite que novas funcionalidades sejam adicionadas para atender a contextos específicos de trabalhos futuros. No entanto, os seguintes recursos já estão disponíveis. Quanto à configuração dos parâmetros de entrada, o simulador é capaz de gerar um número arbitrário de iterações segundo as distribuições *beta*, *gamma*, *normal*, *poisson* e *uniforme*. Com relação às políticas de escalonamento de iterações, o simulador permite a escolha de duas políticas: estática (Static) e dinâmica (Dynamic), ambas presentes no modelo de programação OpenMP [Dagum and Menon 1998]. Na primeira estratégia, as iterações de um laço são particionadas estaticamente antes da execução e são atribuídas às *threads* seguindo uma política *round-robin*. Já na estratégia Dynamic (também conhecida por *Fixed Size Chunk*), as iterações são particionadas em tempo de execução e são atribuídas sob demanda às *threads*. Em ambas as estratégias, o tamanho da partição (*chunk size*) é definido de antemão.

Por fim, o simulador desenvolvido é capaz de gerar um traço detalhado que contém as seguintes informações: (i) número de *threads* simuladas; (ii) quantidade de iterações sintéticas geradas; (iii) tamanho de cada iteração; (iv) iterações atribuídas a cada *thread*; e (v) carga de trabalho total atribuída a cada *thread*. Essas informações são salvas em um arquivo e podem ser utilizadas posteriormente para se realizar um estudo detalhado da simulação, como o apresentado na Seção 5.

4.3. Algoritmo Genético

O simulador desenvolvido permite que as diferentes políticas de balanceamento de cargas existentes sejam estudadas. No entanto, utilizá-lo de maneira isolada dificulta a identificação de políticas alternativas ainda mais eficientes. Uma estratégia adicional, que pode ser adotada em conjunto com a técnica de simulação, é a técnica de exploração do espaço de soluções do problema. Nesse caso, o problema de escalonamento de laços é encarado como um problema de minimização e um algoritmo de busca heurística é a ele aplicado. Esse algoritmo utiliza a função-objetivo de minimização para orientar sua busca e, ao final do processo, encontrar um escalonamento de iterações às *threads* que minimize o valor dessa função.

Diferentes algoritmos de buscas heurísticas são conhecidos. Eles consistem em uma ferramenta básica da Inteligência Artificial e são frequentemente aplicados em diferentes domínios da computação [Coello 1999]. Nesse trabalho, um algoritmo genético foi utilizado para explorar o universo de soluções do problema de escalonamento de laços. Esse algoritmo foi selecionado por dois principais motivos: (i) tem se mostrado promissor em diferentes outros contextos [Konfrst 2004]; e (ii) não exige qualquer conhecimento de como novos estados são gerados a partir de um dado estado. A seguir é feita uma apresentação do funcionamento desse algoritmo e, então, é apresentada uma discussão de como ele foi adaptado para o contexto estudado.

O Algoritmo 1 apresenta o funcionamento, em linhas gerais, de um algoritmo genético. Esse algoritmo recebe dois parâmetros, o tamanho da população (*tampop*) e a quantidade de iterações (*n*), e retorna o estado encontrado que possuir o menor valor avaliado pela função-objetivo. A ideia básica é evoluir uma população de estados ao longo do tempo segundo a teoria da evolução de Darwin. Para tanto, um conjunto de estados iniciais (*pop*) é gerado (linha 2). Nesse conjunto, cada estado (*organismo*) possui duas

[1] `https://github.com/cart-pucminas/scheduler`

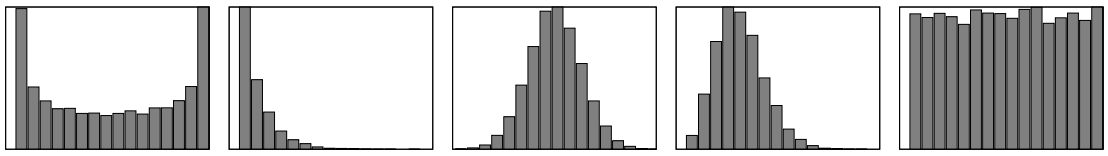
Algoritmo 1 Descrição genérica de um algoritmo genético.

```
 1: function ALGORITMO-GENÉTICO(tampop)
 2:     pop ← POPULAÇÃO-INICIAL(tampop)
 3:     repeat
 4:         novapop ← ∅
 5:         casais ← ∅
 6:         for i from 1 to 2 × tampop do
 7:             casais ← casais ∪ SELEÇÃO(pop)
 8:         for i from 1 to tampop do
 9:             casal ← par aleatório (x, y) ∈ casais
10:             if probabilidade α then
11:                 filhos ← CRUZAMENTO(casal)
12:                 for all filho ∈ filhos do
13:                     if probabilidade β then
14:                         filho ← MUTAÇÃO(filho)
15:             novapop ← novapop ∪ filho
16:         SUBSTITUIÇÃO(pop, novapop)
17:     until critério de parada seja satisfeito
18:     return MELHOR-ORGANISMO(pop)
```

propriedades: a descrição do estado (*cromossomo*), e o valor da função-objetivo para aquele estado (*fitness*). Em seguida, alguns organismos são selecionados para o cruzamento, considerando seu valor de *fitness* (linhas 6 e 7). Organismos com valor de *fitness* mais altos possuem uma maior probabilidade de serem selecionados, possuindo assim uma maior chance de que seus genes perpetuem e a qualidade da população aumente com o passar das gerações. Organismos selecionados para o cruzamento são, em seguida, escolhidos aleatoriamente para formarem pares, cruzarem, com uma alta probabilidade α, e gerarem novos organismos *filhos* (linha 9 a 11). Nesse processo, o cromossomo de um novo organismo é criado trocando-se o material genético entre o par de organismos-pai. Assim que um novo organismo é gerado, ele pode sofrer alguma mutação em seu cromossomo, usualmente com uma pequena probabilidade β (linha 14). Finalmente, para compor a nova população, alguns organismos da população antiga são escolhidos aleatoriamente para serem substituídos pelos novos organismos gerados (linha 16), e então uma nova geração começa. Esse processo termina quando um critério de parada é alcançado (linha 17).

Para o problema de escalonamento de laços, o Algoritmo 1 foi adaptado da seguinte forma: cada cromossomo é um vetor que indica o mapeamento das iterações às *threads* e a *fitness* de cada cromossomo é dada pela função-objetivo definida na Seção 3. Para criar a população inicial, POPULAÇÃO-INICIAL() gera, aleatoriamente, o cromossomo de cada organismo. Os operadores genéticos foram definidos da seguinte forma:

- **SELEÇÃO()**: seleciona indivíduos de acordo com o Algoritmo da Roleta [Goldberg and Deb 1991];
- **CRUZAMENTO()**: gera dois indivíduos e utiliza somente um ponto de corte;
- **MUTAÇÃO()**: altera, aleatoriamente, o valor de um gene de um cromossomo;
- **SUBSTITUIÇÃO()**: aplica a técnica de elitismo [Bhandari et al. 1996] na população antiga e escolhe aleatoriamente os demais organismos que irão compor a nova população, considerando tanto a população antiga remanescente quanto os novos organismos gerados no processo de cruzamento.



(a) *Beta*. $a = 0,5$ e $b = 0,5$ (b) *Gamma*. $a = 1$ e $b = 2$ (c) *Normal*. $\bar{x} = 1$ e $\sigma = 10$ (d) *Poisson*. $\mu = 4$ (e) *Uniforme*. $a = 0$ e $b = 512$

Figura 2. Distribuições e parâmetros considerados.

5. Discussão de Resultados

Nessa seção, são discutidos os resultados obtidos quando a metodologia proposta foi aplicada. Inicialmente, são discutidas as cargas de trabalho, as estratégias de escalonamento consideradas e os parâmetros do algoritmo genético. Em seguida, resultados da convergência do algoritmo genético proposto são apresentados, seguidos pelos resultados de estudo das políticas existentes.

5.1. Parâmetros da Metodologia

As cargas de trabalho foram geradas com base em cinco diferentes funções de distribuição probabilística: *beta*, *gamma*, *normal*, *poisson* e *uniforme*. A Figura 2 apresenta as distribuições e os parâmetros utilizados em cada caso. Para cada uma das cinco cargas de trabalho sintéticas, foram considerados laços de 128, 256 e 512 iterações, totalizando 15 cenários.

Dentre as estratégias de escalonamento existentes, foram utilizadas as estratégias estática (Static) e dinâmica (Dynamic) do OpenMP. Outras estratégias de escalonamento mais robustas são conhecidas [Srivastava et al. 2012], no entanto, elas não foram consideradas uma vez que a estratégia Dynamic se equipara a essas técnicas quando utilizada com blocos (*chunks*) de iterações de tamanho fixo ótimo [Balasubramaniam et al. 2012]. No presente trabalho, ambas as estratégias foram estudadas considerando-se *chunks* de tamanho 1, 4, 8, 16 e 32. Em todos os cenários o número de *threads* foi fixado em 32.

Por fim, os seguintes valores para os parâmetros genéticos foram aplicados: $80\,\%$ de taxa de cruzamento, $10\,\%$ de taxa de mutação, $90\,\%$ de taxa de substituição e $1\,\%$ de taxa de elitismo. Além disso, adotou-se como critério de parada do algoritmo genético quando a melhor solução encontrada se estagna por 1.000 iterações sucessivas. Esses valores interferem no tempo de convergência do algoritmo e foram estabelecidos de maneira empírica. A Seção 5.2 discute o quão adequada foi a escolha desses parâmetros.

5.2. Convergência do Algoritmo Genético

Como dito anteriormente, o algoritmo genético proposto explora o espaço de soluções do problema buscando minimizar o desbalanceamento de carga existente entre um conjunto de *threads* (Equação 1). A Figura 3 apresenta um exemplo da evolução do algoritmo genético (AG) quando uma carga de 512 iterações que seguem a distribuição *beta* é considerada. Partindo de uma solução com um grau de desbalanceamento crítico, o algoritmo gradualmente combina as diferentes soluções de sua população até convergir para uma solução que é $9{,}5\,\mathrm{x}$ e $57{,}6\,\mathrm{x}$ melhor que as soluções encontradas pelas estratégias Dynamic e Static, respectivamente. Um comportamento similar foi observado para os demais cenários.

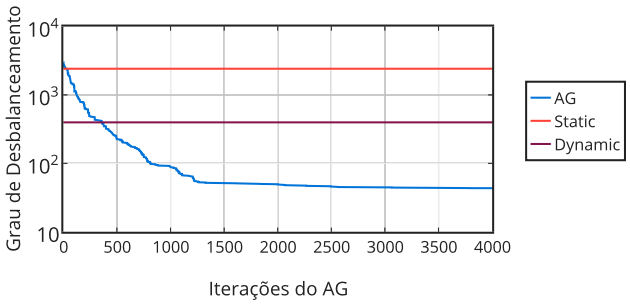


Figura 3. Evolução do AG para uma distribuição *beta* com 512 iterações.

A convergência do algoritmo genético para os demais cenários pode ser concluída a partir das informações compiladas na Tabela 1. Considerando a Equação 1, a solução ótima (teórica) para o problema de escalonamento corresponde a atribuir a cada *thread* uma carga de trabalho igual a carga de trabalho total dividida pelo número de *threads*. Nesse caso, a carga de trabalho atribuída a cada *thread* será a mesma e o desvio padrão relativo observado na carga média de trabalho atribuída será igual a zero.

Tabela 1. Desvio padrão relativo observado na carga de trabalho média.

Número de iterações	*Beta*	*Gamma*	*Normal*	*Poisson*	*Uniforme*
128	6,05 %	12,53 %	1,37 %	7,76 %	3,22 %
256	1,34 %	1,41 %	1,16 %	2,53 %	1,52 %
512	0,26 %	0,22 %	0,73 %	1,68 %	0,68 %

Nos experimentos realizados, o algoritmo genético encontrou, em todos os cenários considerados, uma solução em que a carga de trabalho média atribuída às *threads* coincide com a carga média da solução ótima. No entanto, o desvio padrão relativo observado variou de 0,22 % a 12,53 %, ficando sempre abaixo de 5 % em 12 de 15 casos. Esses resultados apontam dois fatos. Em primeiro lugar, o algoritmo genético proposto converge para uma solução próxima da ótima, validando então seu uso no estudo do problema de escalonamento de laços. Em segundo lugar, estratégias de escalonamento podem fazer uso dessas informações para alcançar soluções ainda mais eficientes.

5.3. Avaliação das Estratégias de Escalonamento

Na Seção 5.2 foram apresentados resultados que indicam uma convergência do algoritmo genético proposto em direção à solução ótima para o problema estudado. A Figura 4 compara o grau de desbalanceamento da solução encontrada pelo algoritmo genético (AG) com as soluções encontradas pelas estratégias Static e Dynamic. As estratégias Static e Dynamic foram executadas com *chunks* de tamanho 1, 4, 8, 16 e 32. Porém, a Figura 4 apresenta somente os melhores resultados em cada caso.

O resultado para a estratégia SRR também é apresentado. Essa estratégia foi projetada a partir de uma análise das soluções encontradas pelo algoritmo genético, que considerou a carga média de trabalho atribuída às *threads* e o desvio padrão observado nos diferentes cenários. Em termos gerais, a ideia dessa estratégia é atribuir em pares as iterações de um laço às *threads* seguindo um esquema *round-robin*, de forma que ao final do processo cada *thread* receba uma carga de trabalho próxima à carga de trabalho média estimada. Para tanto, cada par é composto pelas iterações de maior e menor cargas ainda não atribuídas.

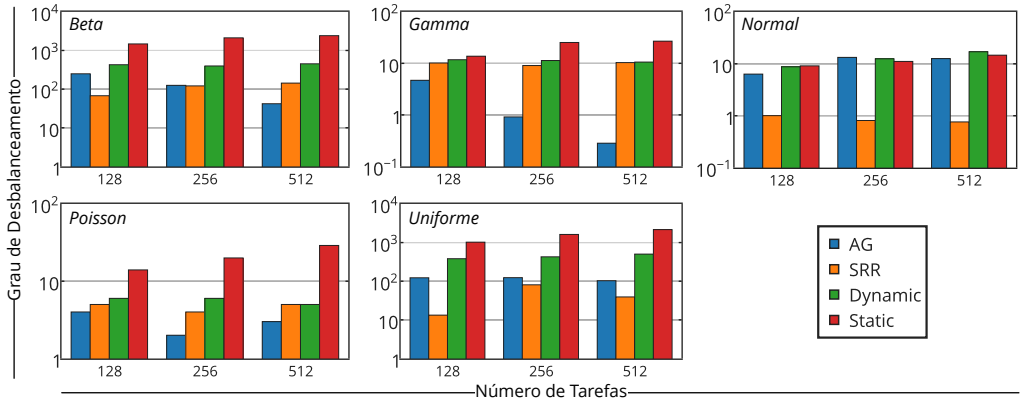


Figura 4. Avaliação das estratégias de escalonamento de laços.

Com base nos resultados é possível verificar que em 14 dos 15 cenários estudados a solução encontrada pelo AG supera às encontradas pelas estratégias existentes (Static e Dynamic), sendo no mínimo $1{,}2\,\mathrm{x}$ melhor (distribuição *normal*, 512 iterações) e no máximo $40{,}2\,\mathrm{x}$ melhor (distribuição *gamma*, 512 iterações). A solução encontrada pelo AG foi superada na distribuição *normal* com 256 iterações pela estratégia Static em $1{,}2\,\mathrm{x}$. Essa situação é melhor detalhada na Seção 5.4.

Diferente das estratégias Static e Dynamic, a estratégia SRR leva em consideração a carga de trabalho e, então, obtém uma solução superior às encontradas por essas duas estratégias em todos os cenários estudados. Esse ganho observado varia de $1{,}1\,\mathrm{x}$ (distribuição *gamma*, 512 iterações) a $32{,}3\,\mathrm{x}$ (distribuição *uniforme*, 512 iterações), sendo expressivo em todas as distribuições, exceto para a distribuição *gamma*. Esse fato é explicado pela natureza da distribuição e relação com a estratégia de escalonamento e será discutido em maiores detalhes a seguir.

5.4. Análise da Distribuição de Cargas

Avaliando as estratégias de escalonamento a partir da Figura 4, três casos típicos podem ser identificados: (i) o algoritmo genético apresenta a melhor solução, seguido pelas estratégia SRR, Dynamic e Static; (ii) o algoritmo genético apresenta a melhor solução, com as estratégias SRR e Dynamic equiparadas, e a estratégia Static em seguida; e (iii) a estratégia SRR se destaca com a melhor solução. Esses três casos podem ser observados para as distribuições *beta*, *gamma* e *normal* quando 512 iterações são consideradas.

A Figura 5 apresenta uma visão detalhada do desempenho do algoritmo genético e das três estratégias de balanceamento nesses cenários, com a linha vermelha horizontal indicando a solução ótima (teórica) para o problema. Para a distribuição *beta* é possível perceber um claro desbalanceamento nas soluções encontradas pelas estratégias Static e Dynamic. Já para as estratégias SRR e AG, observa-se que as soluções encontradas aproximam-se substancialmente da ótima, com a diferença entre as duas soluções explicada pela magnitude do eixo vertical do gráfico.

Para a distribuição *gamma*, é possível constatar o forte desbalanceamento provocado pela estratégia Static e uma solução próxima da ótima encontrada pelo algoritmo genético. No entanto, as estratégias SRR e Dynamic encontram-se equiparadas em um meio termo. No caso da estratégia SRR, o motivo deve-se à própria distribuição e estratégia: como existem poucas iterações grandes, essas iterações são atribuídas às *thre-*

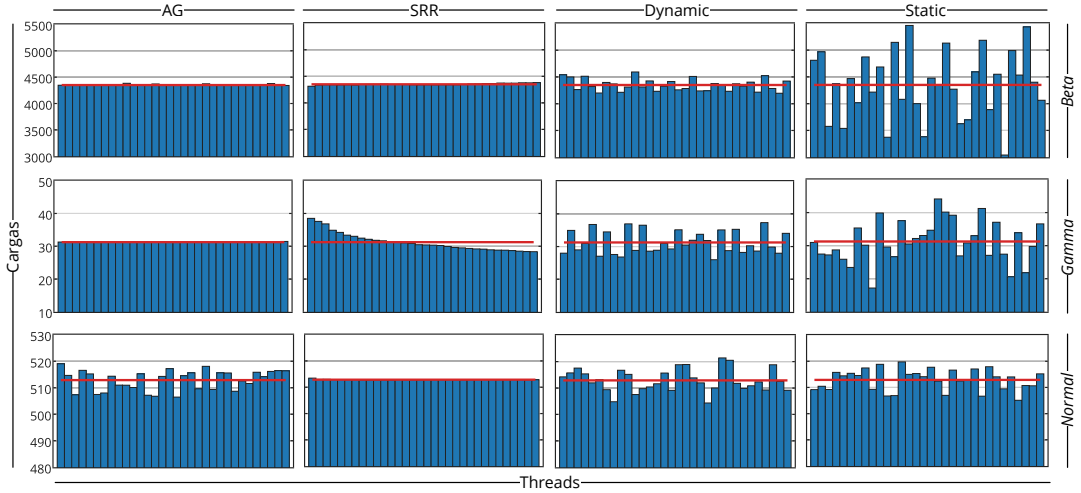


Figura 5. Cargas atribuídas à cada *thread* em execuções com 32 *threads*.

ads inicialmente consideradas no escalonamento, causando um desbalanceamento entre o conjunto de *threads* posteriormente considerado. Esse fato é evidenciado pelos degraus presentes no gráfico. Portanto, a estratégia SRR é mais adequada para distribuições que possuam um número balanceado de iterações com carga grande e pequena.

Por fim, para a distribuição *normal* observa-se que a estratégia SRR se aproxima consideravelmente da solução ótima enquanto as demais estratégias apresentam resultados desbalanceados com valores similares. Nessa situação, observou-se que o algoritmo genético ficou estagnado em um máximo local e, portanto, não convergiu para uma solução ótima. Além disso, a uniformidade existente na quantidade de iterações com carga pequena e grande na distribuição favorece a estratégia SRR. Um comportamento similar foi observado na distribuição *uniforme*.

6. Conclusões

Esse trabalho apresentou uma metodologia que possibilita o projeto e exploração de novas estratégias de escalonamento de laços. Nessa metodologia, a técnica de simulação foi empregada para se estudar as principais políticas de escalonamento existentes e um algoritmo genético foi usado para se explorar o espaço de soluções do problema. Baseado nos resultados obtidos quando essa metodologia foi aplicada, uma nova estratégia, denominada SRR, foi proposta. Os resultados revelaram que as políticas clássicas apontadas na literatura, Static e Dynamic, são de $1{,}2\,\mathrm{x}$ a $40{,}2\,\mathrm{x}$ piores que as soluções ótimas estimadas pelo algoritmo genético em 14 dos 15 cenários estudados. Já a estratégia SRR se mostrou de $1{,}1\,\mathrm{x}$ a $32{,}3\,\mathrm{x}$ melhor que as políticas existentes, em todos os casos considerados. Essa observação valida a metodologia proposta, possibilitando seu uso no projeto de novas estratégias de escalonamento de laços. Como trabalhos futuros, pretende-se empregar a metodologia para guiar o projeto de novas estratégias, aprimorar a política SRR para considerar aspectos relacionados à plataforma e estender a metodologia, acrescentando uma fase adicional que valide as estratégias no ambiente OpenMP.

Agradecimentos

Os autores agradecem o CNPq, a FAPEMIG e o INRIA pelo suporte parcial ao trabalho.

Referências

Balasubramaniam, M., Sukhija, N., Ciorba, F., Banicescu, I., and Srivastava, S. (2012). Towards the scalability of dynamic loop scheduling techniques via discrete event simulation. In *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1343–1351, Shanghai, China.

Bhandari, D., Murthy, C. A., and Pal, S. K. (1996). Genetic algorithm with elitist model and its convergence. *International Journal of Pattern Recognition and Artificial Intelligence*, 10(06):731–747.

Chen, B. and Guo, D. (2014). A static scheduling scheme of multicore compiler for loop load imbalance in openmp. In *International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 1–4, Fujian, China.

Coello, C. A. C. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308.

Dagum, L. and Menon, R. (1998). Openmp: An industry standard api for shared-memory programming. *IEEE Computational Science Engineering*, 5(1):46–55.

Ding, W., Zhang, Y., Kandemir, M., Srinivas, J., and Yedlapalli, P. (2013). Locality-aware mapping and scheduling for multicores. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 1–12, Shenzhen, China.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Morgan Kaufmann.

Hajieskandar, A. and Lotfi, S. (2010). Parallel loop scheduling using an evolutionary algorithm. In *International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 1, pages 314–319, Chengdu, China.

Konfrst, Z. (2004). Parallel genetic algorithms: Advances, computing trends, applications and perspectives. In *IEEE International Parallel Distributed Processing Symposium (IPDPS)*, pages 162–, Santa Fe, New Mexico.

Olivier, S. L., Porterfield, A. K., Wheeler, K. B., Spiegel, M., and Prins, J. F. (2012). Openmp task scheduling strategies for multicore numa systems. *Int. J. High Perform. Comput. Appl.*, 26:110–124.

Patni, J., Aswal, M., Pal, O., and Gupta, A. (2011). Load balancing strategies for grid computing. In *International Conference on Electronics Computer Technology (ICECT)*, volume 3, pages 239–243, Kanyakumari, India.

Skiena, S. S. (2008). *The Algorithm Design Manual*. Springer, 2nd edition.

Srivastava, S., Malone, B., Sukhija, N., Banicescu, I., and Ciorba, F. (2013). Predicting the flexibility of dynamic loop scheduling using an artificial neural network. In *IEEE International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 3–10, Bucharest, Romania.

Srivastava, S., Sukhija, N., Banicescu, I., and Ciorba, F. (2012). Analyzing the robustness of dynamic loop scheduling for heterogeneous computing systems. In *International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 156–163, Munich, Germany.

Sukhija, N., Malone, B., Srivastava, S., Banicescu, I., and Ciorba, F. (2014). Portfolio-based selection of robust dynamic loop scheduling algorithms using machine learning. In *IEEE International Parallel Distributed Processing Symposium Workshops (IPDPSW)*, pages 1638–1647, Phoenix, USA.

Tantawi, A. N. and Towsley, D. (1985). Optimal static load balancing in distributed computer systems. *J. ACM*, 32(2):445–465.